网络空间安全创新创业实践

The birth attack of SM3

实验报告

王祥宇---202000460053

项目时间：2022 年 7 月 7 日

目录

一、项目任务

- Project: implement the naïve birthday attack of reduced SM3

二、实验过程及思路

【一】SM3 算法的实现过程

根据参考文献完成 SM3 算法的实现，具体思路和参数数值在文献中均详细给出，故在此只给出大体思路。

首先根据文献设置初始值（文献 4.1）：

```
#4.常数与函数
#4.1初始值
iv='7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e'
```

然后依次实现常数 T、布尔函数和置换函数（文献 4.2　4.3　4.4）：（其中的循环左移函数提前实现）

```
#4.2常数
def T(j):
    if j<16:
        return 0x79cc4519
    return 0x7a879d8a

#4.3布尔函数
def FF(x,y,z,j):
    if j<16:
        return x^y^z
    return (x&y)|(y&z)|(z&x)
def GG(x,y,z,j):
    if j<16:
        return x^y^z
    return (x&y)|(~x&z)

#4.4置换函数
def P0(x):
    return x^CycleLeft(x, 9)^CycleLeft(x, 17)
def P1(x):
    return x^CycleLeft(x, 15)^CycleLeft(x, 23)
```

然后是算法具体实现，文献给出 SM3 的基本概述：明文 m 长度小于 64bit，经过消息填充和迭代压缩最后输出长度为 256bit 的杂凑值。

首先实现消息填充 fill（文献 5.2）：（m 为十六进制表示，故其中的有些数值是文献的 1/4）

```
#5.2填充
def fill(m):#均以十六进制为单位
    a=len(m)*4
    m=m+'8'
    k=112-(len(m)%128)
    m=m+'0'*k+'{:016x}'.format(a)
    return m
```

然后实现消息扩展 extend（文献 5.3.2）：

```
#5.3迭代压缩
#5.3.2消息扩展
def extend(x):
    w=[]
    for i in range(16):
        w.append(int(x[i*8:(i+1)*8],16))
    for j in range(16,68):
        w.append(P1(w[j-16]^w[j-9]^\
                    CycleLeft(w[j-3], 15))^CycleLeft(w[j-13], 7)^w[j-6])
    for j in range(68,132):
        w.append(w[j-68]^w[j-64])
    return w
```

压缩函数 CF:

```
#5.3.3压缩函数CF
def CF(vi,bi):
    w=extend(bi)
    a,b,c,d,e,f,g,h=int(vi[0:8],16),\
                    int(vi[8:16],16),\
                    int(vi[16:24],16),int(vi[24:32],16),\
                    int(vi[32:40],16),int(vi[40:48],16),\
                    int(vi[48:56],16),int(vi[56:64],16)
    for j in range(64):
        ss1=CycleLeft((CycleLeft(a,12)+e+CycleLeft(T(j),j))%pow(2,32),7)
        ss2=ss1^CycleLeft(a,12)
        tt1=(FF(a,b,c,j)+d+ss2+w[j+68])%pow(2,32)
        tt2=(GG(e,f,g,j)+h+ss1+w[j])%pow(2,32)
        d=c
        c=CycleLeft(b,9)
        b=a
        a=tt1
        h=g
        g=CycleLeft(f,19)
        f=e
        e=P0(tt2)
    abcdefgh=int('{:08x}'.format(a)+\
                 '{:08x}'.format(b)+'{:08x}'\
                .format(c)+'{:08x}'.format(d)\
                +'{:08x}'.format(e)+'{:08x}'.\
                format(f)+'{:08x}'.format(g)+\
                '{:08x}'.format(h),16)
    return '{:064x}'.format(abcdefgh^int(vi,16))
```

有了压缩函数 CF，最后实现迭代过程 iteration 和输出杂凑值（文献 5.3.1 5.4）：

```
#5.3.1迭代过程
def iteration(b):
    n=len(b)
    v=iv
    for i in range(n):
        v=CF(v,b[i])
    return v

#5.4杂凑值
def sm3hash(m):#m为16进制串
    m=fill(m)
    b=grouping(m)
    return iteration(b)
```

【二】SM3 的生日攻击实现过程

birth_bit（x）

首先输入参数 x 表示寻找 x 比特的碰撞，这里先规定 x 必须为 4 的倍数，便于后续和十六进制进行转换，输出的杂凑值为十六进制。寻找的碰撞长度为 x 比特，所以一共有 2^x 种可能的杂凑值，故 ans 数组大小为 2^x。ans 数组的下标表示杂凑值，值表示 hash 值的原像。

因为生日攻击的复杂度为 1.17*sqrt（n），所以需要对 1 到 2^128 之间的数（即消息）进行遍历。令 temp 等于消息杂凑值的整数形式，如果 ans[temp]=-1，说明原来没有消息的杂凑值的整数形式等于 temp；相反，如果 ans[temp]不等于 -1，说明发现了碰撞，完成任务。具体代码如下图：

```
#寻找x比特长度的碰撞
def birth_bit(x):#x必须为4的倍数
    ans=[-1]*(2**x)
    y=x/2
    p=2**y
    #print(p)
    for i in range(2**128):#生日攻击的复杂度为1.17*s
        temp=int(sm3hash(hex(i))[0:int(x/4)],16)#要注
        if ans[temp]==-1:#如果数组值发生了改变，说明
            ans[temp]=i;
        else:
            print(i,ans[temp])#ans数组的下标表示hash
```

birth_30bit（x）

但不妨换个思路，上面规定输入的 x 必须为 4 的倍数，是因为 hash 值是十六进制，方便截取前 x 比特。有可能会出现如下情况：杂凑值的第七位（十六进制）相同，第八位一个为 a，一个为 b，此时虽然运用上个函数是判断无碰撞的，但是 a 和 b 的二进制表示第一位相同，所以用多了一位碰撞。所以对上个函数进行了改进。

由于输入 x=32 时 python 会出现如下错误（x=31 也会出这个错误）：

```
Traceback (most recent call last):
  File "D:\桌面\the birth attack of SM3.py", line 44, in <module>
    birth(32)
  File "D:\桌面\the birth attack of SM3.py", line 33, in birth
    ans=[-1]*(2**x)
MemoryError
```

可能是因为内存不够。

那 30bit 的碰撞呢？当然 29bit 也是有可能的，但为了追求更长的碰撞在此以 30bit 碰撞为例。所以该函数中基本上没有参数，目的就是为了寻找 30bit 的碰撞。大体思路和 ans 数组的设置与上个函数相同。不同主要在 temp 上，我们只需先将十六进制的 hash 值转化为二进制，然后截取[2:32]即可。实现代码如下：

```
#寻找x比特长度的碰撞
def birth_30bit(x):#x必须为4的倍数
    ans=[-1]*(2**x)
    y=x/2
    p=2**y
    #print(p)
    for i in range(2**128):#生日攻击的复杂度为1.17*sqrt(n)
        temp=int(bin(int(sm3hash(hex(i))[0:8],16))[2:x+2],
        if ans[temp]==-1:#如果数组值发生了改变，说明原来至
            ans[temp]=i;
        else:
            print(i,ans[temp])#ans数组的下标表示hash值，值
```

三、实验结果及检验

在上述两个函数中我们只求找到两个碰撞即可，所以当运行的时候会发现有很多对都满足比如说 28bit 的碰撞，而且会有重复的对，在此不在考虑如何将重复的对归到一起，只求找出一对碰撞即可。

第一个碰撞函数 birth_bit（x），由于内存原因，x 最大为 28。将 x 输入为 28，得到如下结果：

```
--
33175 30844
40192 17934
41207 17079
```

我们对第一组进行检验，调用 sm3hash 输入 33175 和 30844：

```
print(sm3hash(hex(33175)))
print(sm3hash(hex(30844)))
```

结果如下：

```
--
3af58c7262a5737fe818718c61e35ef086f62b4ee6be60a9c47517678168906a
3af58c73380f89480d57768f22cee4ef4b350cb3eda0367b9349e2565426a8d1
```

会发现十六进制杂凑值的前 7 位确实是的相同的。其实第八位 2 和 3 的二进制表示前两位也相同，所以这一对共有 30bit 的碰撞。

第二个函数输入 x 为 30，结果如下所示：

```
41207 17079
48989 22354
50417 39085
72525 13910
73508 40747
73543 41977
```

这里仍是以第一组为例，进行检验：

```
print(sm3hash(hex(41207)))
print(sm3hash(hex(17079)))
```

结果如下:

85a263bec2a573043cd561edf6dc64077cf143165df57687428c5f3f8b9a9991
85a263bf2f1c76908c4168420fbde4c146ce289d90b372ed71490289e11d99fb

会发现前 7 位仍然是一样的，第八位是 e 和 f，二进制表示的前三位也一样。所以我们本意是寻找 30bit 的碰撞，但很幸运地找到了 31bit 的碰撞。

最后结果:

41207 和 17079 找到 31bit 的碰撞。

四、实验反思与总结

本次项目首先实现了 SM3 算法，个人感觉 SM3 算法要比 SM2 算法就编程实现上要简单很多，但从参考文献的难易程度也能看出。在实现生日攻击的过程中首先寻找十六进制的碰撞，后来发现可能十六进制不一样但仍存在相同的比特位，故进行了改进，又很幸运的多找了一位的碰撞。最终找到了 31 比特的碰撞。通过此次实现将学过的生日攻击运用了实践中，加深了对生日攻击和杂凑函数碰撞的理解，为以后的密码学习奠定了基础。

五、参考文献

【SM3 算法中的每一步操作均可以在下面的文献中找到】

http://www.sca.gov.cn/sca/xwdt/2010-12/17/1002389/files/302a3ada057c4a73830536d03e683110.pdf